Project Vision

A compact, Raspberry Pi–based guitar rig that lets players:

- **Play silently** via headphones or studio monitors with low-latency amp + effects.
- **Feed an amp** as a pedal/amp modeler.
- **Capture clean DI** (per-string when available) for re-amping.
- **Auto-generate tablature** from real-time playing, with per-string tracking via a **hexaphonic pickup** (or a custom string sensor), exportable to common tab formats.

Core User Stories

1. **Silent Jamming**: Plug guitar → headphone mix with amp/cab sim + pedals. Save/load presets.
2. **Pedalboard Mode**: Guitar → Pi → line out → amp/FRFR cab. Footswitchable scenes.
3. **DI Capture & Re-amp**: Record raw input (mono, or 6-channel with hex pickup). Later play DI back through the same rig or external amp.
4. **Live Tab**: See scrolling tab while playing. After a take, clean it up, quantize, and export.

Hardware (Initial + Expandable)

Minimum MVP

- **Compute**: Raspberry Pi 4B or 5 (prefer 5 for headroom). Active cooling.
- **Storage**: 32–128GB fast microSD or USB SSD (preferred for audio reliability).
- **Audio I/O**: Initially using a **Focusrite Scarlett 2i2** (class-compliant USB 2-in/2-out) with Hi-Z instrument input and headphone out. Direct monitor **off** (handled in software).
- **Power**: Stable, low-noise PSU; consider USB power isolation to reduce ground noise.
- **Foot I/O (optional)**: USB MIDI footswitch or GPIO foot buttons for preset changes and looping.

Hex Pickup / Multi-String Expansion (v2)

- **Pickup**: Decided on **Roland GK-3** hexaphonic pickup.
- **Breakout**: Custom-built 13-pin breakout box to provide 6 separate string outputs for audio interface input.
- **Interface**: A **6–8 channel** class-compliant USB interface will be used to capture the discrete string signals from the breakout.

Software Stack

- **OS**: Raspberry Pi OS (64-bit). Enable **low-latency** kernel/audio tuning.
- **Audio Backend**: JACK or ALSA with minimal buffer sizes (target 48 kHz, 64–128 frames).

- **DSP Host**: **Carla** will be the primary LV2 host. It offers a full-featured GUI with patchbay graph, plugin scanning, preset management, and JACK integration—making it the most user-friendly choice for prototyping on the Pi.
- **UI**: **Qt/QML** front-end (touch-friendly). Large toggles, meters, preset browser, tab editor.
- **Data**:
- Audio: 24-bit/48 kHz WAV (mono DI or 6ch for hex).
- Projects: YAML/JSON session files (routing, presets, tab, tempo).
- Tab export: **Guitar Pro (GPX)** and **MusicXML**; MIDI optional.

Real-Time Signal Flow

Live (MVP): Guitar (Hi-Z) → ADC ch1 → [Noise Gate] → [Drive] → [Amp Sim] → [Cab IR] → [EQ/Comp] → DAC out 1–2 (Headphones/Monitors) + optional Line Out.

Record & Re-amp:

- Record **pre-FX DI** to disk while monitoring **post-FX**.
- For re-amp: Route DI from disk → same chain (or physical re-amp box → real amp). Capture mic'd amp back in.

Hex Mode (v2):

- 6 discrete inputs → per-string onset/pitch/technique detection → tab engine (polyphonic aware).
- Mixdown for FX path: per-string DSP or summed bus depending on CPU budget.

Latency Budget Targets

- **ADC/DAC + driver**: \~2–6 ms (low-latency kernel, 64–128 buffer at 48 kHz).
- **DSP**: \~1–4 ms (depends on IR length and models).
- **UI & misc**: \~<1 ms incremental.
- **End-to-end goal**: **<10 ms** round trip; stretch goal **\~6–8 ms**.

Tactics: prefer shorter IRs (or zero-latency filters) live; use longer IRs offline or with CPU headroom. Pin threads to cores, disable CPU scaling, isolate cores for audio.

Effects & Amp Modeling Options

- **Quick wins**: Gate, EQ, compressor, delay, chorus, tremolo, reverb (algorithmic).

- **Amp/Cab**: Basic analog-style models or **IR-based cabs**.

- **IR Placement**: IR loaders simulate the cab + mic response, so they sit **after the amp sim/distortion stage**. In practice: Amp Sim → Cab IR → EQ/Comp → FX.

- **DSP Cost**: IRs are convolution processes handled in the DSP engine. Shorter IRs (128–512 taps) = lower latency but less accuracy; longer IRs (1024–2048+ taps) = more realistic but higher CPU and added latency.

- **Advanced**: Neural Amp models (NAM/ONNX) if CPU allows; fallback to efficient tube-sim filters when CPU is constrained.

Tablature Detection (Design)

Assumption: Hex pickup provides one channel per string → simplifies polyphonic pitch detection.

1. **Per-String Preprocess**: HPF \~60–80 Hz on low strings, LPF to reduce pick noise; normalize.
2. **Onset Detection**: Energy + spectral flux with adaptive thresholds per string.
3. **Pitch Tracking**: YIN / McLeod MPM / autocorrelation hybrid; 48 kHz → decimate for speed if needed.
4. **Fret Mapping**: Convert frequency → semitone vs. known tuning; pick **nearest fret** with intonation tolerance. Handle bends by continuous pitch curves → notated as bend arrows.
5. **Technique Heuristics**:
6. **Hammer-on/Pull-off**: Onset without strong pick transient + smooth pitch transition.
7. **Slides**: Continuous gliss between semitones.
8. **Vibrato**: Low-depth periodic pitch modulation.
9. **Palm-mutes**: Envelope + high-freq damping profile.
10. **Rhythm/Quantization**: Global tempo estimate + beat grid; allow manual tap-tempo. Post-take quantize with user-set swing/strength.
11. **Chord Handling**: Simultaneous onsets across multiple strings → stacked notes in tab.
12. **Error Correction UI**: After capture, an **inline tab editor** to fix ambiguous notes, shift string assignments, and nudge rhythms.

Fallback (MVP without hex): Mono DI → polyphonic pitch is hard; constrain to **single-note mode** or “one string at a time” training to demo concept.

UI Outline (Qt/QML)

- **Home**: Big input meter, tuner, latency indicator, quick preset tiles.
- **Rig View**: Node graph or pedal chain with draggable order. Save/recall presets.
- **Record**: Arm DI/Bus, take list, loop capture, re-amp routing.
- **Tab**: Live scrolling staff/tab; post-take editor with piano-roll-style timing grid.
- **Settings**: Buffer size, sample rate, hex mapping (string → input), pickup gain trims, footswitch mapping.

File/Project Structure

```
/Projects/
  MySong/
    MySong.session.json       # routing, presets, tempo
    Takes/
      take_001_DI.wav         # mono (or 6ch)
      take_001_bus.wav        # post-FX reference (optional)
    Tabs/
      take_001.gpx
      take_001.musicxml
    Presets/
      Crunchy.yaml
```

Engineering Plan

Phase 0 – Lab Setup (1–2 days)

- Install OS; enable low-latency settings (USB IRQ affinity, CPU governor performance).
- Verify round-trip latency with loopback on the Scarlett 2i2; log best buffer sizes at 48 kHz. Expect \~8–10 ms at 64-sample buffer and \~11–13 ms at 128-sample buffer.

Phase 1 – MVP Audio (1–2 weeks)

- Stand up **LV2 host** (Carla/MOD-style) with a simple chain: gate → EQ → light drive → cab IR → limiter.
- UI: Meters, preset save/load, latency readout.
- Record clean DI while monitoring post-FX.

Phase 2 – Re-amp & Looping (1 week)

- Playback DI through chain; A/B live vs. DI.
- Add simple looper (fixed length) and metronome.

Phase 3 – Tab (mono prototype) (1–2 weeks)

- Per-note onset + pitch (single-note mode) → tab line.
- Export MusicXML/GPX for verification in external editors.

Phase 4 – Hex Mode (2–4+ weeks)

- 6-channel input wiring and calibration.
- Per-string trackers + chord onset grouping.
- Technique heuristics; editor for corrections.

Phase 5 – Migration & Portability

- If needed, replace parts of the LV2 chain with **custom C++ DSP** for tighter latency and per-string control.
- Preset management, footswitching, scene morphs.
- Consider JUCE/Superpowered abstraction to prepare for cross-platform (desktop/mobile).

Bill of Materials (example picks)

- Raspberry Pi 5 + heatsink/fan, 5V/5A PSU.
- USB SSD (250–500GB) or high-endurance microSD.
- Class-compliant USB 2×2 interface with Hi-Z input and headphone out.
- Headphones/monitors; optional DI/re-amp box for real amp workflows.
- (v2) Hex pickup + 6–8ch USB interface or dedicated hex breakout; shielded cabling.
- USB MIDI footswitch (optional).

Risks & Mitigations

- **Latency/CPU**: Favor efficient DSP; cap IR lengths; use integer math where possible; pin threads.
- **Noise/Grounding**: Use proper Hi-Z, consider USB isolators, star-ground strategy.
- **Polyphonic Tab Accuracy**: Provide a fast edit loop; allow training profiles per guitar/pickup.
- **Thermals**: Active cooling; monitor throttling.

Nice-to-Have Add-Ons

- **Practice tools**: Backing track player, slowdown/loop AB, smart key detection.
- **Mobile control**: Web UI for preset switching from phone.
- **Cloud preset/tab sync** (opt-in).

Portability & Future Hardware

Software Portability

- **UI Layer (Qt/QML):** Platform-agnostic; keep it UI-only. No DSP logic in QML.
- **Engine Boundary:** Talk to DSP via an API facade (signals/slots or IPC/OSC). Swap Carla/LV2 for custom DSP without breaking UI.
- **Host Flexibility:** Carla/JACK on Pi; later, swap to custom engine (JUCE, RtAudio, PortAudio) with the same external API.
- **Plugins:** Prefer LV2 for ARM/Linux. Wrap critical blocks as internal modules so you can migrate later.
- **Backends:** Abstract JACK/ALSA now; later map to CoreAudio/WASAPI/ASIO with JUCE.
- **Files:** Use standard formats (WAV/JSON/YAML/GPX/MusicXML).

Hardware Integration Paths

- **MVP path:** Re-use a class-compliant **USB Audio Class 2.0 interface** inside the box. Bring its jacks to your panel.
- **OEM path:** Use an **XMOS UAC2 reference design** or similar to integrate a module.
- **Performance path:** Custom PCB with **I²S/TDM codec** (TI/Cirrus/AKM) for lower latency and direct multi-channel hex inputs.
- **Design practices:**
- Balanced outs + headphone amp; Hi-Z preamps with pad/protection.
- Proper clocking, power isolation, analog/digital ground strategy.
- Plan for EMC/FCC/CE compliance.

Architecture Guidance

- Split process: `ui` (Qt app) ↔ `engine` (Carla headless or custom).
- Use a **preset/graph schema** in JSON/YAML so it rehydrates across hosts.
- Express per-string routing in config, not code.

Next Steps

- Scaffold the **Qt/QML UI shell** + minimal DSP and a **single-note tab demo**.